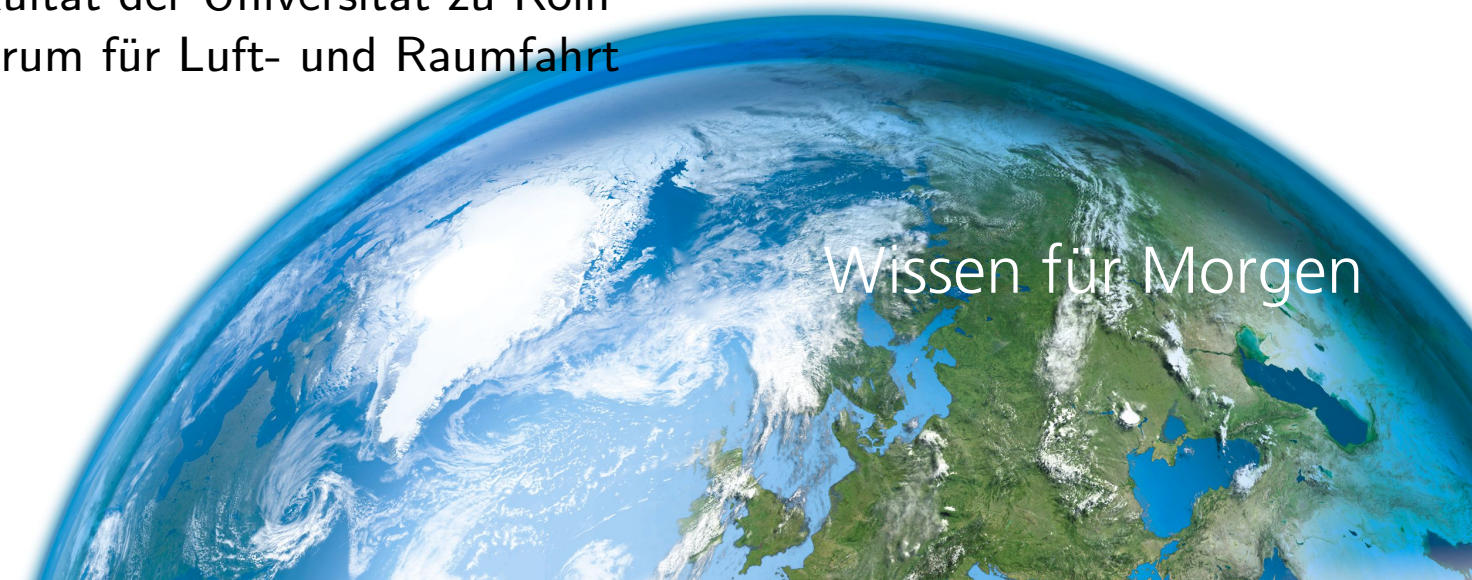
Machbarkeitsstudie des L-BFGS Verfahrens für das Training von Deep Learning Problemen

Ilona Shonia
Erstgutachter: Prof. Dr. Axel Klawonn
Betreuer: Dr. Martin Siggel, Dr. Philipp Knechtges, Janine Weber
Math.-Nat. Fakultät der Universität zu Köln
Deutsches Zentrum für Luft- und Raumfahrt

14.11.2018



DLR

Gliederung

1. Motivation
2. Grundlagen
3. Nichtlineare Optimierung
4. Numerische Ergebnisse
5. Fazit & Ausblick

Motivation
Deep Learning

Deep Learning Methoden, wie künstliche neuronale Netze, sind ein aktives Forschungsgebiet wegen

- dem Durchbruch in der Rechenkapazität moderner Computer
- der Zunahme der zur Verfügung stehenden Trainingsdaten

Das Training von großen Modellen erfordert

- hohe Rechenleistung
- großen Speicherplatz

an die (gradientenbasierten) Optimierungsalgorithmen, die zum Lernen verwendet werden.

Um diesen Aufwand zu reduzieren, werden Gradienten von einem zufällig gewählten Teil des Datensatzes gebildet. Dies gibt dem berechneten Gradienten einen stochastischen Charakter.

Motivation
Problemstellung

Die stochastische Abwandlung des gradientenbasierten Optimierungsalgorithmus Gradient Descent wird in ML erfolgreich verwendet und ist als Stochastic Gradient Descent (SGD) bekannt.

Können jedoch anspruchsvollere Optimierungsalgorithmen unter Verwendung rauschhaltigen Gradienten noch bessere Ergebnisse liefern?

Diese Arbeit beschäftigt sich mit der Anwendbarkeit der L-BFGS Methode mit stochastischen Gradienten für das Training von Deep Learning Probleme.

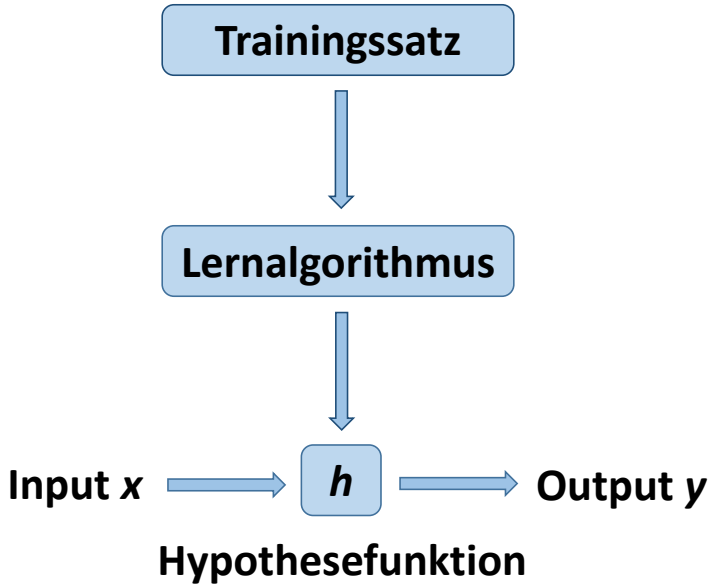
Motivation

Anwendungsproblem

Klassifizierung von Bildern auf Basis des MNIST-Datensatzes.
Ein Problem des überwachten Lernens.

Abbildung: Der MNIST Datensatz
`https://en.wikipedia.org/wiki/MNIST_database`

Maschinelles Lernmodell
Überwachtes Lernen



- Trainingssatz: Datensatz aus Input-Output Paaren $\{(x_i, y_i)\}$, $i = 1, ..., n$, wobei $x_i \in \mathcal{X}$, $y \in \mathcal{Y}$ und $\mathcal{X} \subset \mathbb{R}^k$, $\mathcal{Y} \subset \mathbb{R}^m$ ist.
- Lernalgorithmus: hier neuronale Netze
- Hypothesefunktion: eine Abbildung $h : \mathcal{X} \to \mathcal{Y}$

Maschinelles Lernmodell
Überwachtes Lernen

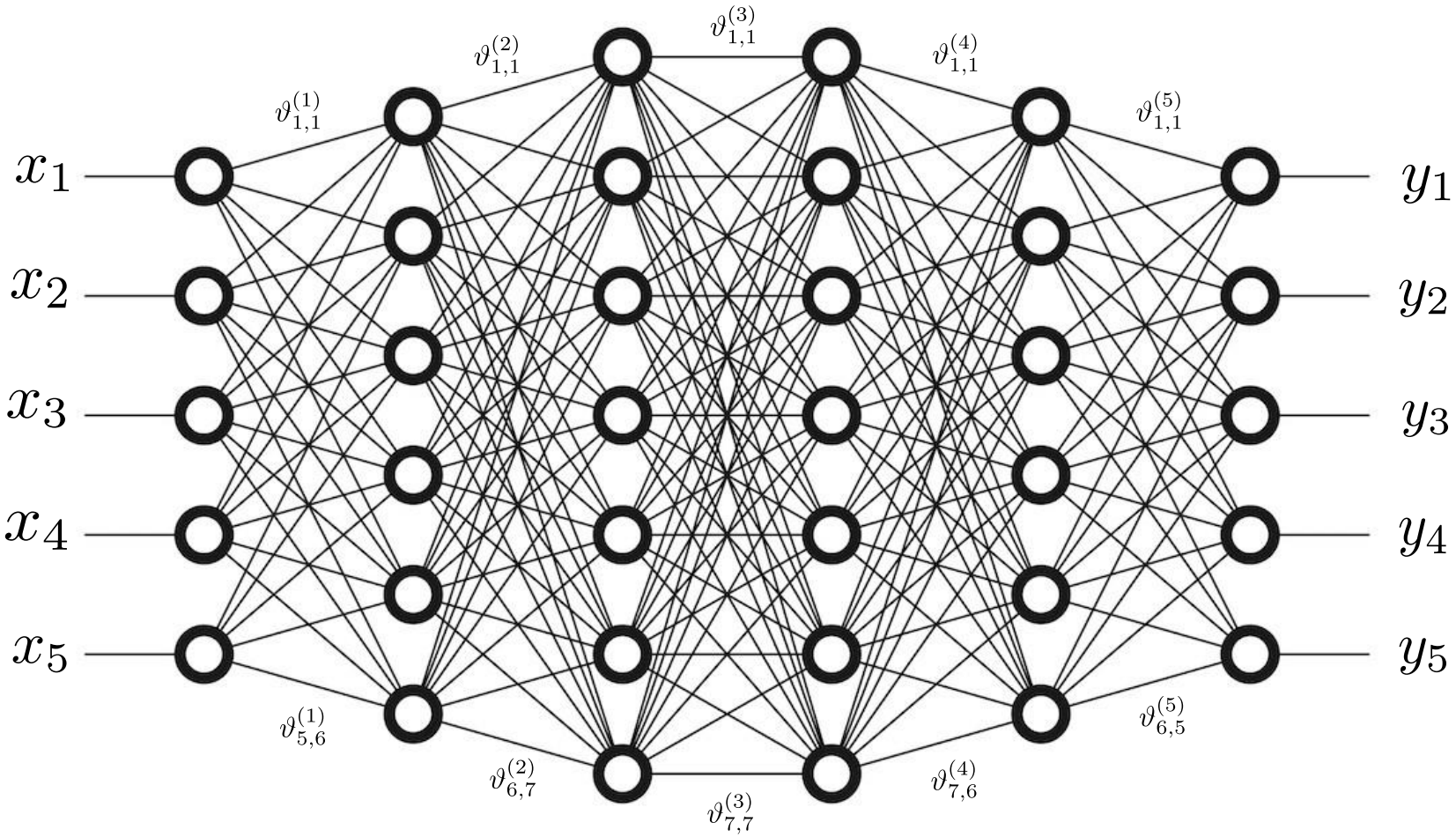
- Modell hängt von den Modellparametern ϑ
- Ziel des Lernens: für jeden neuen Input x eine gute Vorhersage über den Output y zu treffen $\Rightarrow h_\vartheta(x) \approx y$.
- Bewertung der Modellqualität: Fehlerfunktion $J(\vartheta)$, die den Unterschied zwischen der Grundwahrheit y und der von dem Modell gemachten Vorhersage $h_\vartheta(x)$ misst.

Ein einfaches Beispiel der Fehlerfunktion:

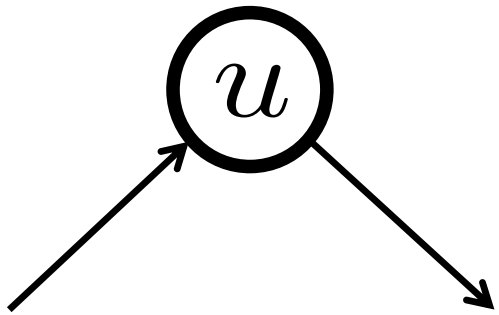
$$J(\vartheta) = \frac{1}{n} \sum_{i=1}^{n} \left(h_\vartheta(x_i) - y_i\right)^2 .$$

Jedoch ist die Darstellung der Fehler- und Hypothesefunktion sehr modellabhängig.

Lernalgorithmen
Künstliche neuronale Netze



Lernalgorithmen
Künstliche neuronale Netze



$$in_u = \sum_{v \in U} \vartheta_{v,u} out_v \qquad out_u = f_{act}(\sum_{v \in V_u} \vartheta_{v,u} out_v + b_u)$$

Lernalgorithmen
Künstliche neuronale Netze

- Hypothesefunktion: Output der letzten Schicht
- Aktivierungsfunktion: z.B. Sigmoid-Funktion

$$f_{act}(x) = \frac{1}{1 + \mathrm{e}^{-x}}$$

- Fehlerfunktion: Kreuzentropie

$$J(\vartheta) = -\frac{1}{n} \sum_{i=1}^{n} \underbrace{\sum_{j} \{y_{i,j} \ln h_{\vartheta,j}(x_i) + (1 - y_{i,j}) \ln (1 - h_{\vartheta,j}(x_i))\}}_{=:J_i(\vartheta)},$$

wobei $y_{i,j} = 1$ ist, falls x_i zu Klasse j gehört und sonst 0.

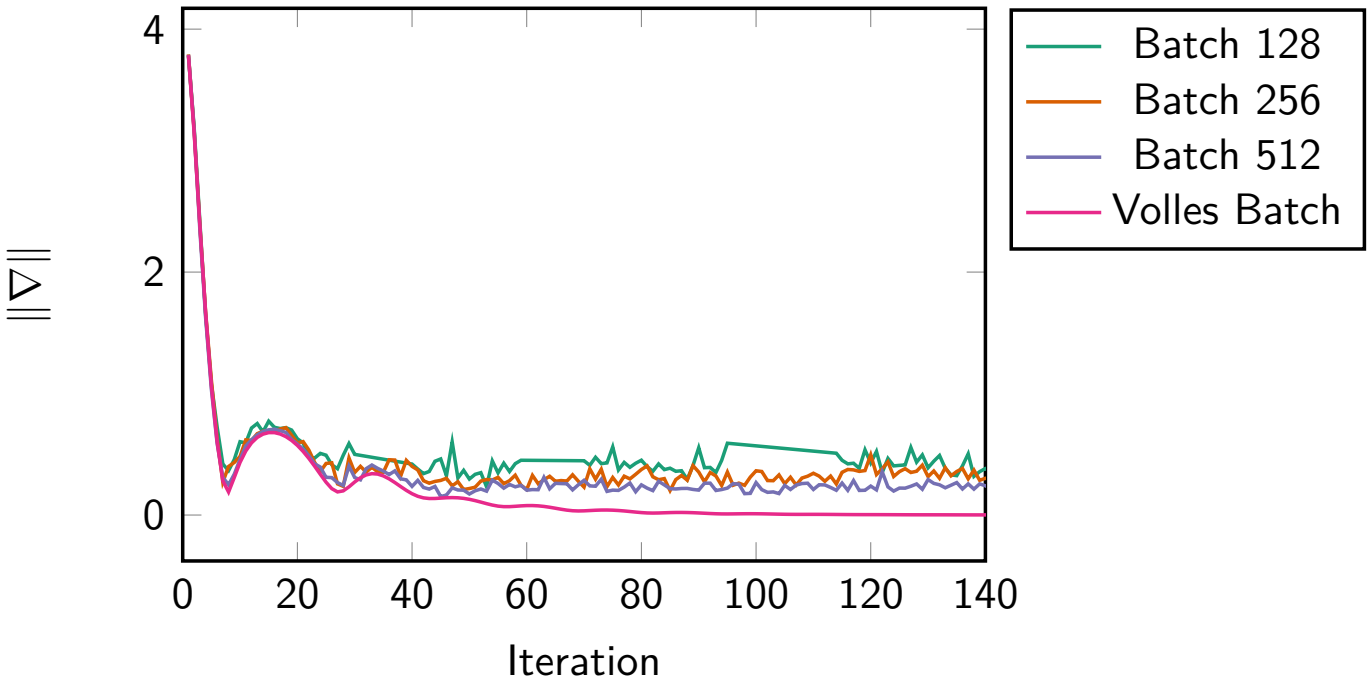
- Training: $min_{\vartheta \in \mathbb{R}^d} J(\vartheta)$

$\Rightarrow$ ein Optimierungsproblem.

Gradientenbasierte Optimierung
Stochastischer Gradient

$$\hat{\nabla} J(\vartheta) = \frac{1}{|S|} \sum_{i \in S} \nabla J_i(\vartheta),$$

wobei S eine Teilmenge des Datensatzes (also ein Batch) ist.



Gradientenbasierte Optimierung
Gradient Descent Varianten

- Das (Batch-)Gradientenverfahren:

$$\vartheta_{k+1} = \vartheta_k - \alpha \nabla J(\vartheta_k) \tag{1}$$

- Das stochastische Gradientenverfahren:

$$\vartheta_{k+1} = \vartheta_k - \alpha \nabla J_i(\vartheta_k) \tag{2}$$

- Das Mini-Batch Gradientenverfahren:

$$\vartheta_{k+1} = \vartheta_k - \alpha \hat{\nabla} J(\vartheta_k) \tag{3}$$

$$\hat{\nabla} J(\vartheta_k) = \frac{1}{|S_k|} \sum_{i \in S_k} \nabla J_i(\vartheta_k)$$

Gradientenbasierte Optimierung
Adam - der Goldstandard in Deep Learning

Algorithmus 1 Adam

Input: Zielfkt. $J(\vartheta)$, Startwert $\vartheta_0, \beta_1, \beta_2 \in [0, 1)$, Schrittgröße α

Initialisiere: Vektoren für die ersten und zweiten Momente:

$m_0 \leftarrow 0$, $v_0 \leftarrow 0$; Iterationszähler: $k \leftarrow 0$

while Abbruchbedingung nicht erfüllt **do**

$k \leftarrow k + 1$; $g_t \leftarrow \hat{\nabla} J(\vartheta_{k-1})$

Aktualisiere verzerrte approximierte Momente des Gradienten:

$m_k \leftarrow \beta_1 m_{k-1} + (1 - \beta_1) g_t$, $v_k \leftarrow \beta_2 v_{k-1} + (1 - \beta_2) g_t^2$

Berechne korrigierte Approximationen an den Momenten des Gradienten:

$\hat{m}_k \leftarrow m_k / (1 - \beta_1^k)$, $\hat{v}_k \leftarrow v_k / (1 - \beta_2^k)$

Setze $\vartheta_k \leftarrow \vartheta_{k-1} - \alpha \hat{m}_k / (\sqrt{\hat{v}_k} + \varepsilon)$

end while

Methoden zweiter Ordnung
Newton-Verfahren

- Das einfachste Gradientenverfahren besitzt nur eine lineare Konvergenzgeschwindigkeit (Theorem 3.3, Nocedal).

- Eine quadratische Konvergenzgeschwindigkeit kann mit dem Newton-Verfahren erreicht werden (Theorem 3.7, Nocedal).

- Iterationsvorschrift für das Newton-Verfahren:

$$\vartheta_{k+1} = \vartheta_k - H_k^{-1} \nabla J_k \tag{4}$$

Nachteil des Newton-Verfahrens: Verwendung der vollen inversen Hesse Matrix $\Rightarrow$ für große Probleme sehr rechen- und speicheraufwändig.

Methoden zweiter Ordnung
Quasi-Newton Verfahren

Methoden aus der Quasi-Newton Klasse versuchen die inverse Hesse Matrix zu approximieren.

- Iterationsvorschrift für allgemeine Quasi-Newton Verfahren:

$$\vartheta_{k+1} = \vartheta_k - \alpha_k B_k \nabla J_k \tag{5}$$

- Das BFGS Verfahren. Approximation der Hesse Matrix durch

$$B_{k+1} = \left(I - \rho_k s_k y_k^T\right) B_k \left(I - \rho_k y_k s_k^T\right) + \rho_k s_k s_k^T, \tag{6}$$

wobei $\rho_k = 1/y_k^T s_k$, $s_k = \vartheta_{k+1} - \vartheta_k$ und $y_k = \nabla J_{k+1} - \nabla J_k$.

Nichtlineare Optimierung
L-BFGS Verfahren

- Das Limited-Memory BFGS (L-BFGS) Verfahren:

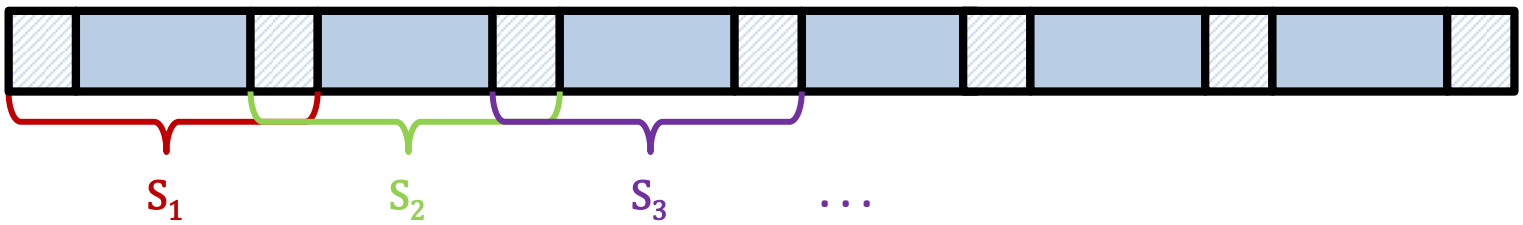
  Berücksichtigung der Krümmungsinformation nur aus den letzten m Iterationen zur Berechnung der Approximation der Hesse Matrix.

  Vorteil: gesparter Speicherplatz. Die approximierte Hesse Matrix wird nie explizit konstruiert und gespeichert, sondern es wird direkt die Suchrichtung $-B_k \nabla J_k$ berechnet.

  Modifizierung der Methode für stochastische Fälle
  $\Rightarrow$ das MB-LBFGS und das PB-LBFGS Verfahren.

Stochastische Quasi-Newton-Verfahren
L-BFGS Verfahren

- Konstruktion von Batches fester Größe in jeder Iteration.
- Erzwingen von Überlappungen zwischen aufeinanderfolgenden Batches.
- Auswertung von stochastischen Gradienten auf den Überlappungen für Berechnung der Änderung im Gradienten.



Stochastische Quasi-Newton-Verfahren
PB-LBFGS

- Anpassung der Batchgröße in jeder Iteration mit dem IPQN-Test

$$\frac{Var_{i \in S_k^v}\left((g_k^i)^T B_k^2 g_k^{S_k}\right)}{|S_k|} \leq \theta^2 \| B_k g_k^{S_k} \|^4$$

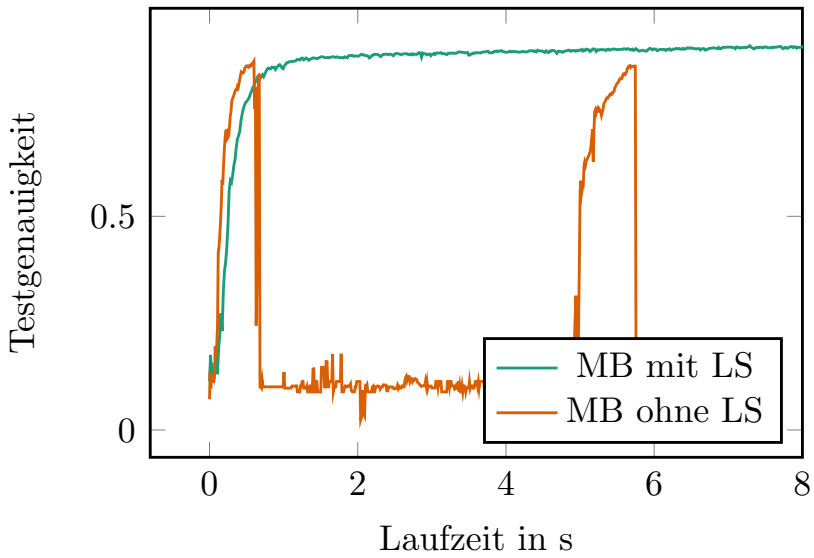
- Stochastische lineare Suche mit dem Startwert

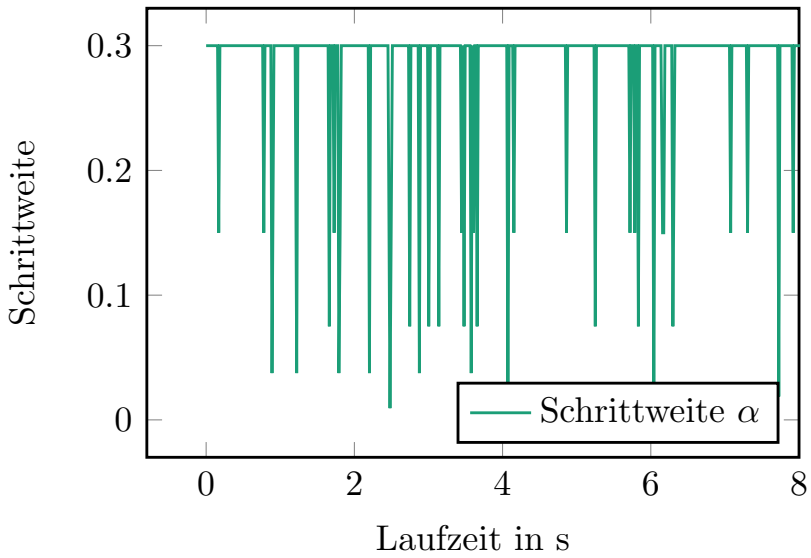
$$\alpha_k = \left(1 + \frac{Var_{i \in S_k^v}\left\{g_k^i\right\}}{|S_k| \| g_k^{S_k} \|^2}\right)^{-1}$$

Numerische Ergebnisse
Das verwendete neuronale Netz

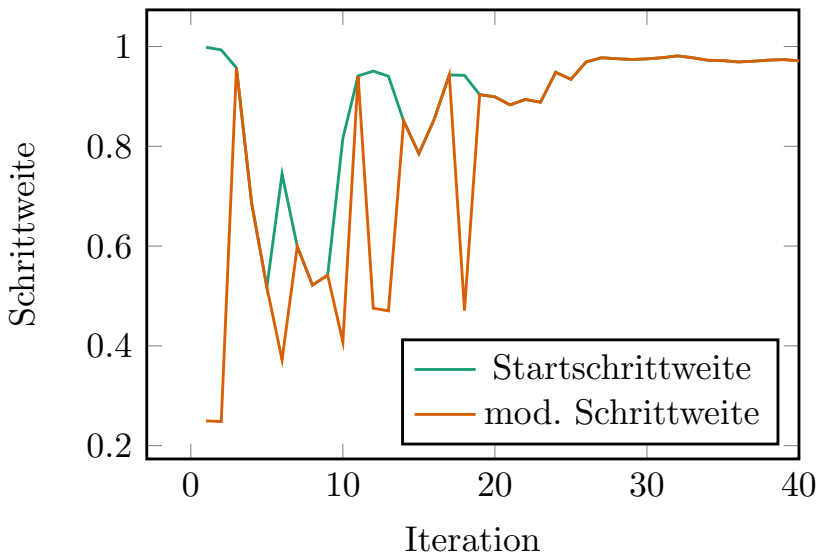
- Input-Layer mit 784 Neuronen (entspricht der Anzahl der Pixeln 28×28).
- Eine verdeckte Schicht mit 25 Neuronen.
- Output-Layer mit 10 Neuronen für die 10 Ziffer $0, 1, ..., 9$.
- Jeweils ein Bias-Unit in der Input- und verdeckten Schicht.
- Insgesamt also $(784 + 1) \cdot 25 + (25 + 1) \cdot 10 = 19\,885$ trainierbare Parameter im Netz.
- Aktivierungsfinktion: Sigmoid-Funktion;
  Fehlerfunktion: Kreuzentropie.

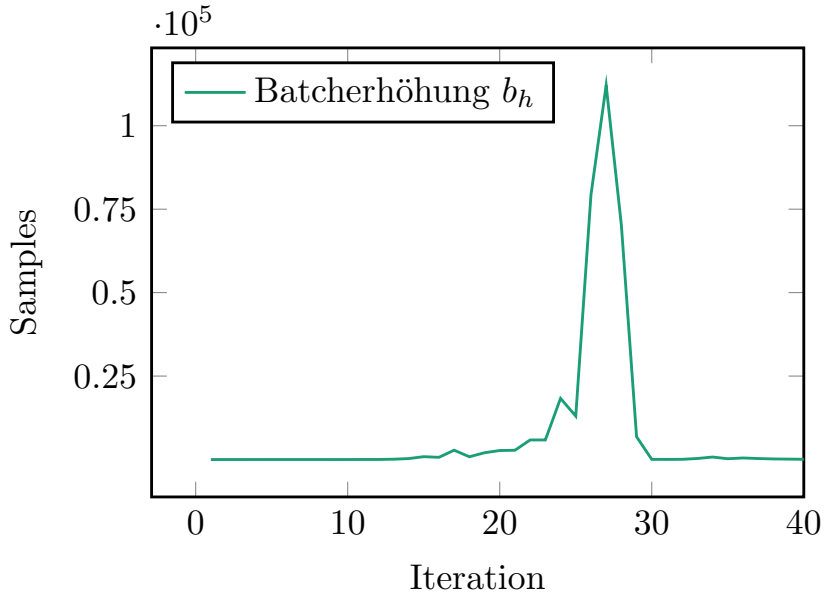
Die Rolle der linearen Suche
MB-LBFGS



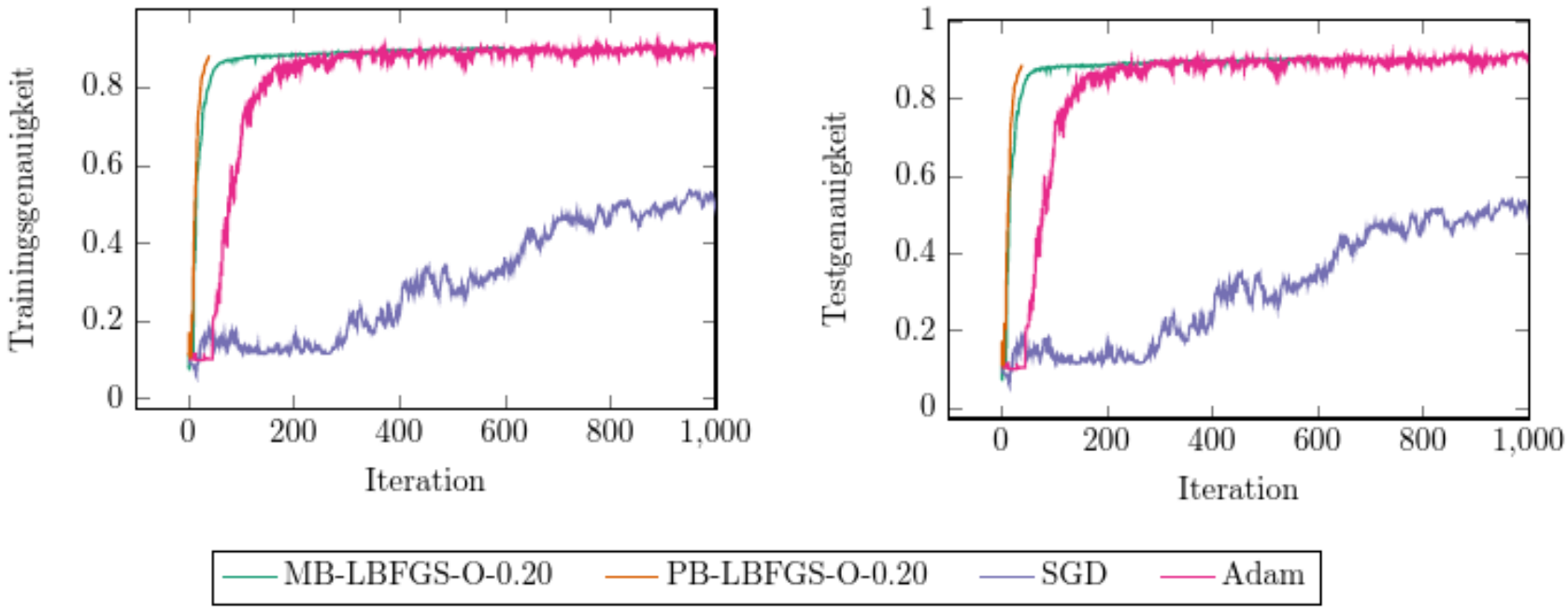


Die Rolle der linearen Suche
PB-LBFGS

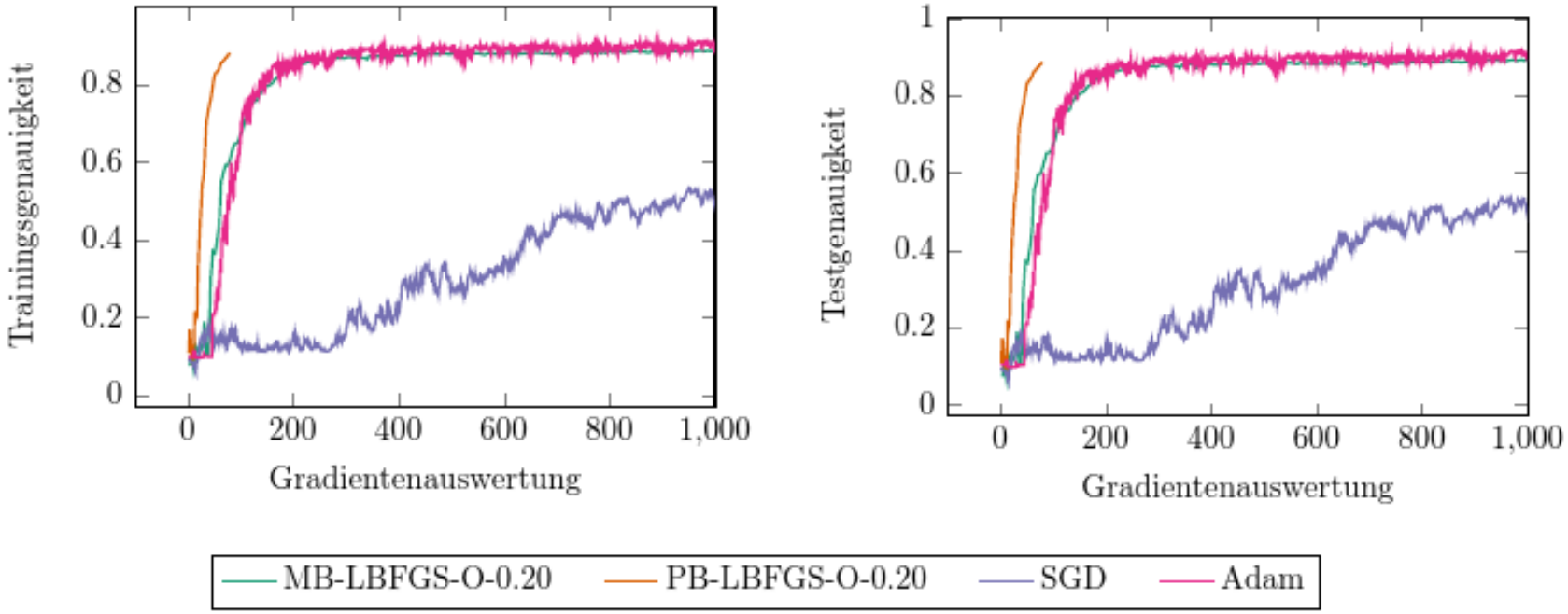




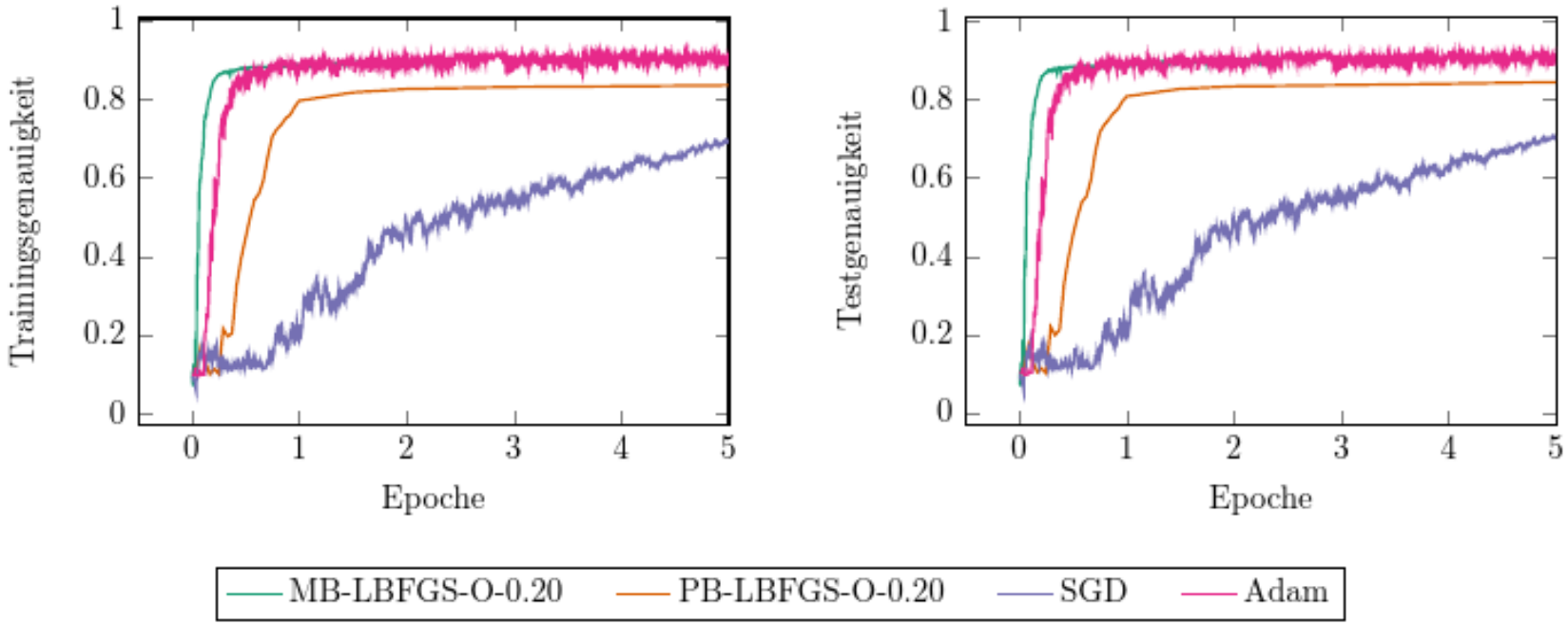
Vergleich der Algorithmen
Iterationen vs Genauigkeit



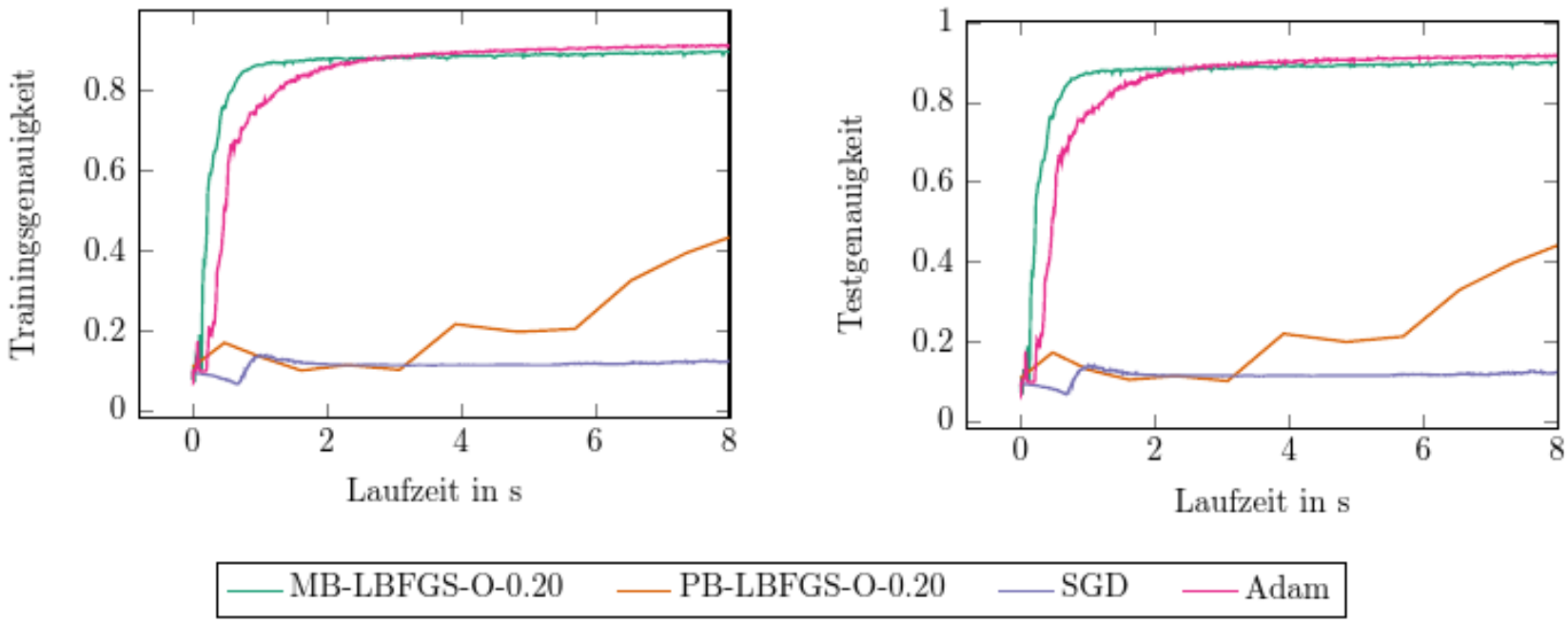
Vergleich der Algorithmen
Gradientenauswertungen vs Genauigkeit



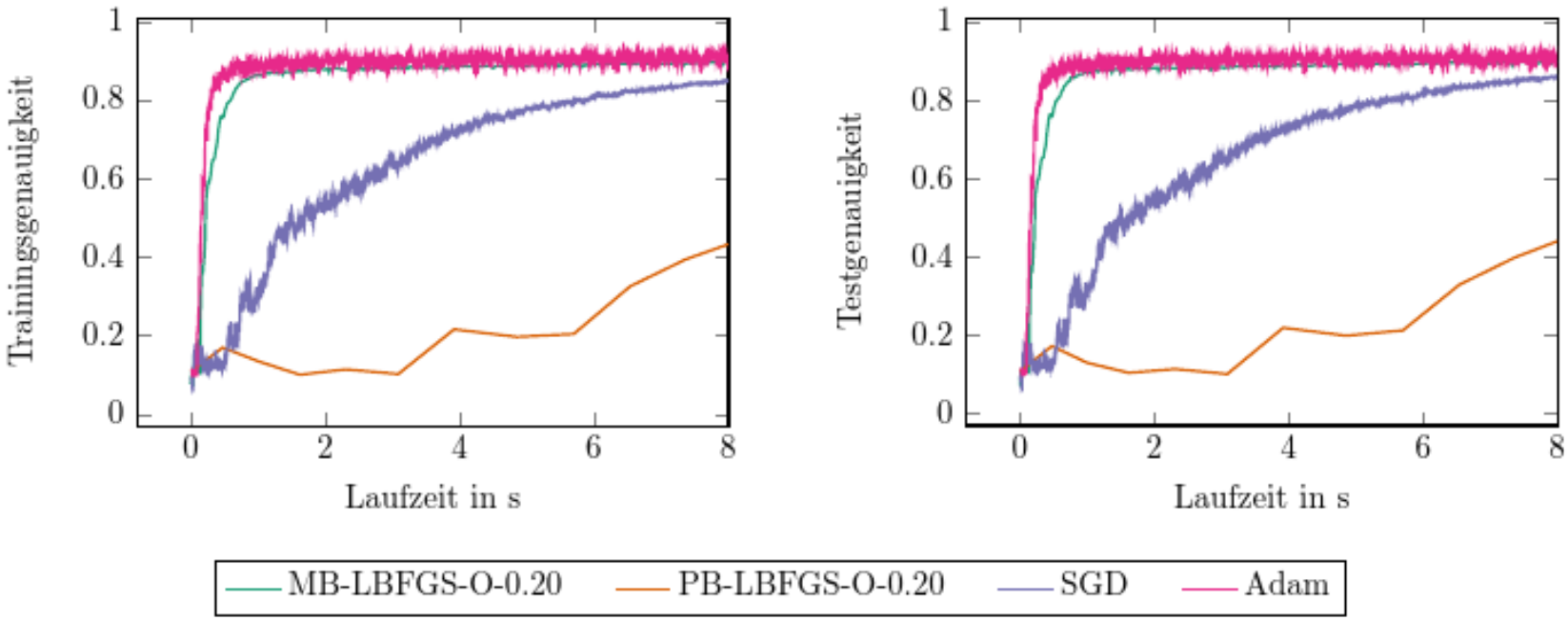
Vergleich der Algorithmen
Epochen vs Genauigkeit



Vergleich der Algorithmen
Laufzeit vs Genauigkeit



Vergleich der Algorithmen
Laufzeit vs Genauigkeit



Fazit & Ausblick

- Das Einbeziehen der approximierten Informationen zweiter Ordnung hilft dabei, **bessere Suchrichtungen** zu konstruieren.
- Dadurch benötigen die Methoden zweiter Ordnung deutlich **weniger Iterationen**, als die von der ersten Ordnung.
- Jedoch **wächst** dabei der **Rechenaufwand** so sehr, dass es nicht möglich ist die Performanz des Adam-Algorithmus in Laufzeit zu übertreffen.
- Das **Design** des neuronalen Netzes und die Wahl der **Hyperparameter** spielen wichtige Rolle bei der Optimierung.
- Für die Zukunft lässt sich hoffen, dass durch Zulassen von **mehr Rauschen in Varianz** bessere Varianten der PB-LBFGS Methode erscheinen.

Fragen

Algorithmus 2 Multi-Batch L-BFGS Methode (MB-LBFGS)

Input: Startwert ϑ_0, Trainingsdatensatz T, Gedächtnisparameter $m \in \mathbb{Z}_{>0}$, Batchgröße r, Größe der Überlappung $o \in (0, 1)$

$k \leftarrow 0$

Generiere initiales Batch S_0

for $k = 0, 1, 2, \ldots$ **do**

Berechne die Suchrichtung $p_k = -B_k g_k^{S_k}$, wähle $\alpha_k > 0$

Berechne $\vartheta_{k+1} = \vartheta_k + \alpha_k p_k$

Generiere nächstes Batch S_{k+1}

Berechne $s_{k+1} = \vartheta_{k+1} - \vartheta_k$ und $y_{k+1} = g_{k+1}^{O_k} - g_k^{O_k}$

Ersetze das älteste Paar (s_i, y_i) durch (s_{k+1}, y_{k+1}) falls bereits m Paare gespeichert, ansonsten füge das neue Paar einfach hinzu

end for

Stochastische Quasi-Newton-Verfahren
PB-LBFGS - Abschätzung der Varianz

$$|\overline{S}_k| \geq \frac{Var_{i \in S_k^v}\left((g_k^i)^T B_k^2 g_k^{S_k}\right)}{\theta^2 \| B_k g_k^{S_k} \|^4} =: b_k$$

$$\rightarrow Var_{i \in S_k^v}\left((g_k^i)^T B_k^2 g_k^{S_k}\right) = \frac{\sum_{i \in S_k^v}\left((g_k^i)^T B_k^2 g_k^{S_k} - \| B_k g_k^{S_k} \|)^2\right)^2}{|S_k^v| - 1}$$

$$\rightarrow Var_{i \in S_k^v}\left\{g_k^i\right\} = \frac{1}{|S_k - 1|} \sum_{i \in S_k^v} \| g_k^i - g_k^{S_k} \|^2$$